



Crypto for Java Developers

Java Forum Stuttgart 2014

Dominik Schadow - bridgingIT

„Crypto is complex and full of backdoors...“



A6

Sensitive Data Exposure

```

graph LR
    TA[Threat Agents] --- AV[Attack Vectors]
    AV --- SW[Security Weakness]
    SW --- TI[Technical Impacts]
    TI --- BI[Business Impacts]
  
```

The flowchart illustrates the progression of a security threat. It starts with 'Threat Agents' (represented by a stick figure icon), which leads to 'Attack Vectors' (represented by a white box with a grey arrow icon). This leads to 'Security Weakness' (represented by a white box with a green arrow icon). Finally, it leads to 'Technical Impacts' (represented by a cylinder icon) and then to 'Business Impacts' (represented by a white box).

| Application Specific | Exploitability DIFFICULT | Prevalence UNCOMMON | Detectability AVERAGE | Impact SEVERE | Application / Business Specific |
|---|--|--|---|--|------------------------------------|
| Consider who can gain access to your sensitive data and any backups of that data. This includes the data at rest, in transit, and even in your customers' browsers. Include both external and internal threats. | Attackers typically don't break crypto directly. They break something else, such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's browser. | The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm usage is common, particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty detecting server side flaws due to limited access and they are also usually hard to exploit. | Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive data such as health records, credentials, personal data, credit cards, etc. | Consider the business value of the lost data and impact to your reputation. What is your legal liability if this data is exposed? Also consider the damage to your reputation. | |

There is a pattern in most data breaches...



1. All passwords are safe!
2. Stolen, but securely hashed!
3. OK, some have a simple hash...
4. Change your password now!

No secret, no key -> no security

~~BaseEncoding.base64().encode(password)~~



passwords security attack random hashes keystore signatures decryption management certificates public transport sensitive configuration pair numbers hybrid session ciphertext information symmetric

private asymmetric digital data id keys authENTICATION dictionary encryption exchange certificate brute-force verification



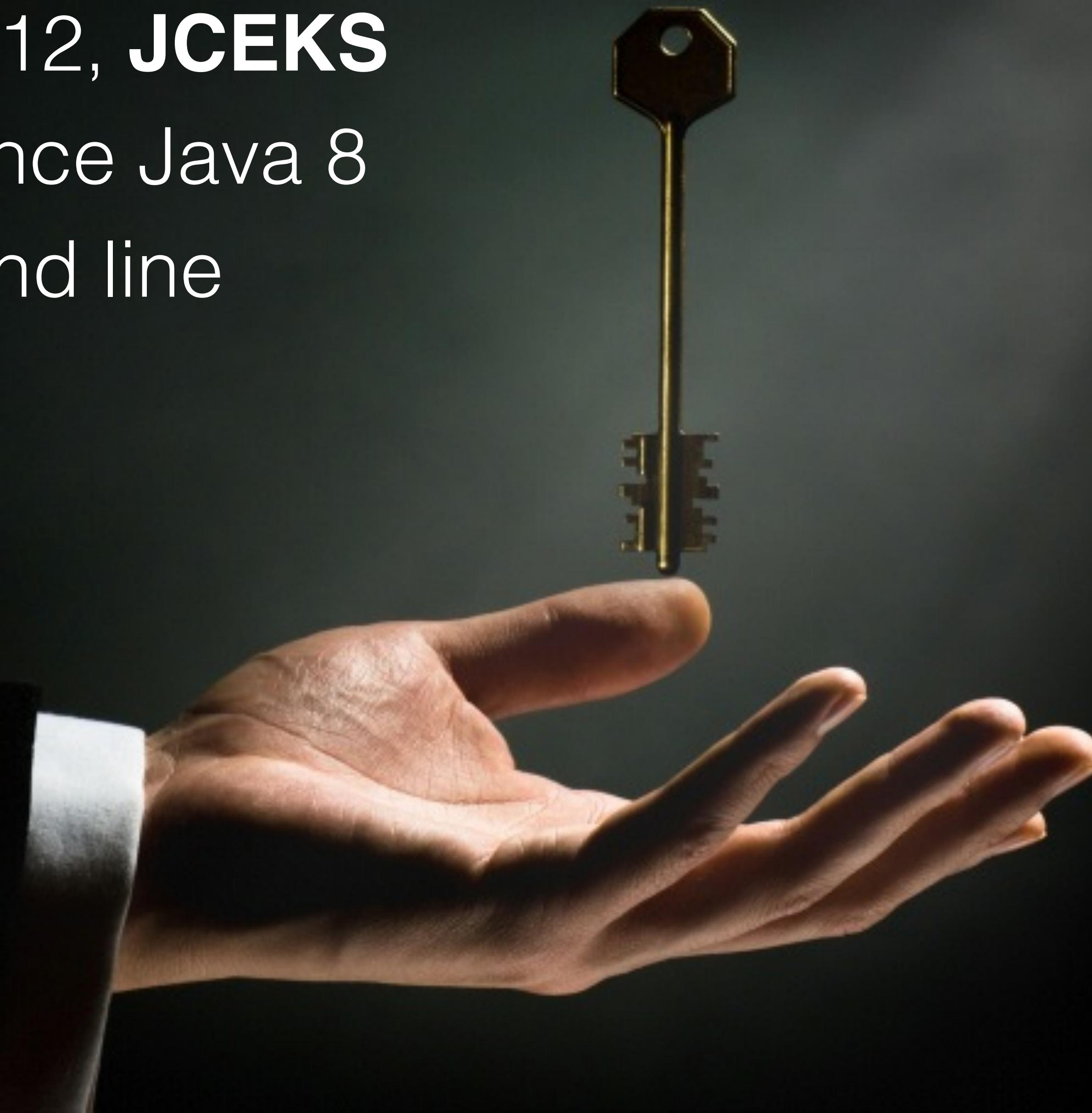
**Pure Java
Spring Configuration
Encryption & Signatures
Hashes**

Pure Java



Java KeyStore stores key pairs, secret keys and certificates

- Common types: PK12, **JCEKS**
- Store passwords since Java 8
- KeyTool on command line



Easy keystore management with KeyStore Explorer

The screenshot shows the KeyStore Explorer application window titled "samples.ks - KeyStore Explorer 5.0.1". The interface includes a toolbar with various icons for file operations and keystore management, and a main table view displaying the contents of the "samples.ks" keystore.

The table has the following columns:

| T | L | E | Entry Name | Algorithm | Key Size | Certificate Expiry | Last Modified |
|----------|-----------|-------------|-----------------------|-----------|----------|-------------------------|-------------------------|
| key icon | lock icon | unlock icon | asymmetric-sample-dsa | DSA | 1024 | 01/Mai/2015 09:54:43... | 01/Mai/2014 09:54:58... |
| key icon | lock icon | unlock icon | asymmetric-sample-rsa | RSA | 2048 | 01/Mai/2015 09:54:07... | 01/Mai/2014 09:54:18... |
| key icon | lock icon | unlock icon | - symmetric-sample | AES | 192 | - | 01/Mai/2014 09:53:34... |

At the bottom of the application window, a status bar displays the information: "KeyStore Type: JCEKS, Size: 3 Entries, Path: '/Users/'".

One shared secret key for symmetric encryption

- ❑ Difficult key exchange
- ❑ Fast



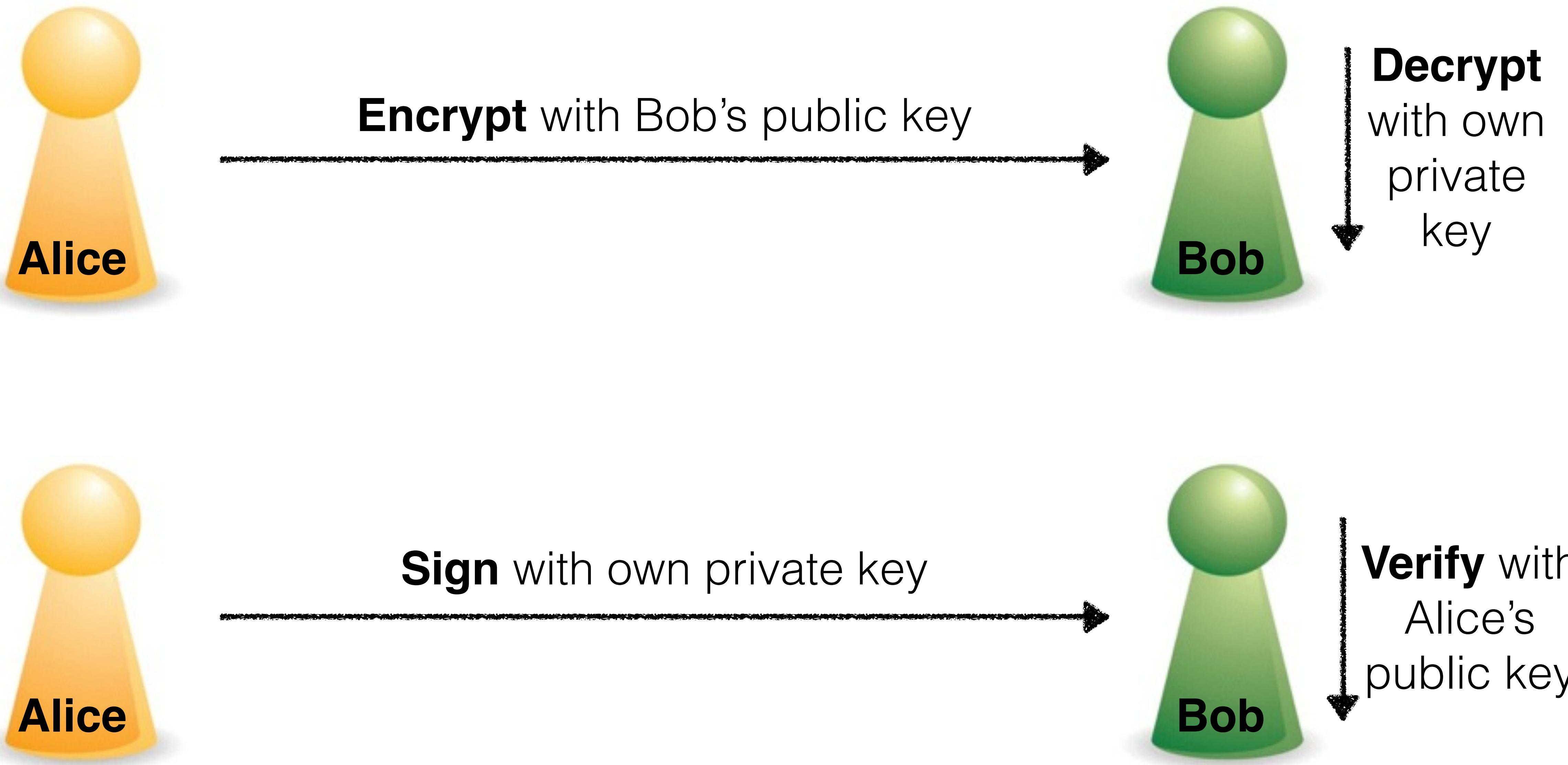
Individual public and private key for asymmetric encryption

- Simple key exchange
- Slower



Hybrid encryption
Content: Symmetric
Key: Asymmetric

Encrypt with counterparts public key, sign with own private key



Demo

(Crypto) frameworks make crypto easier, but come with limitations

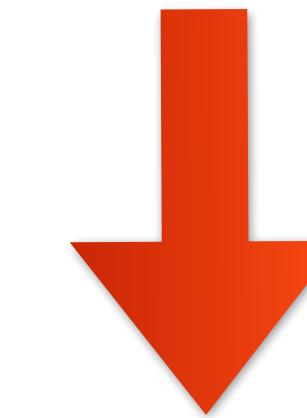


Spring Configuration



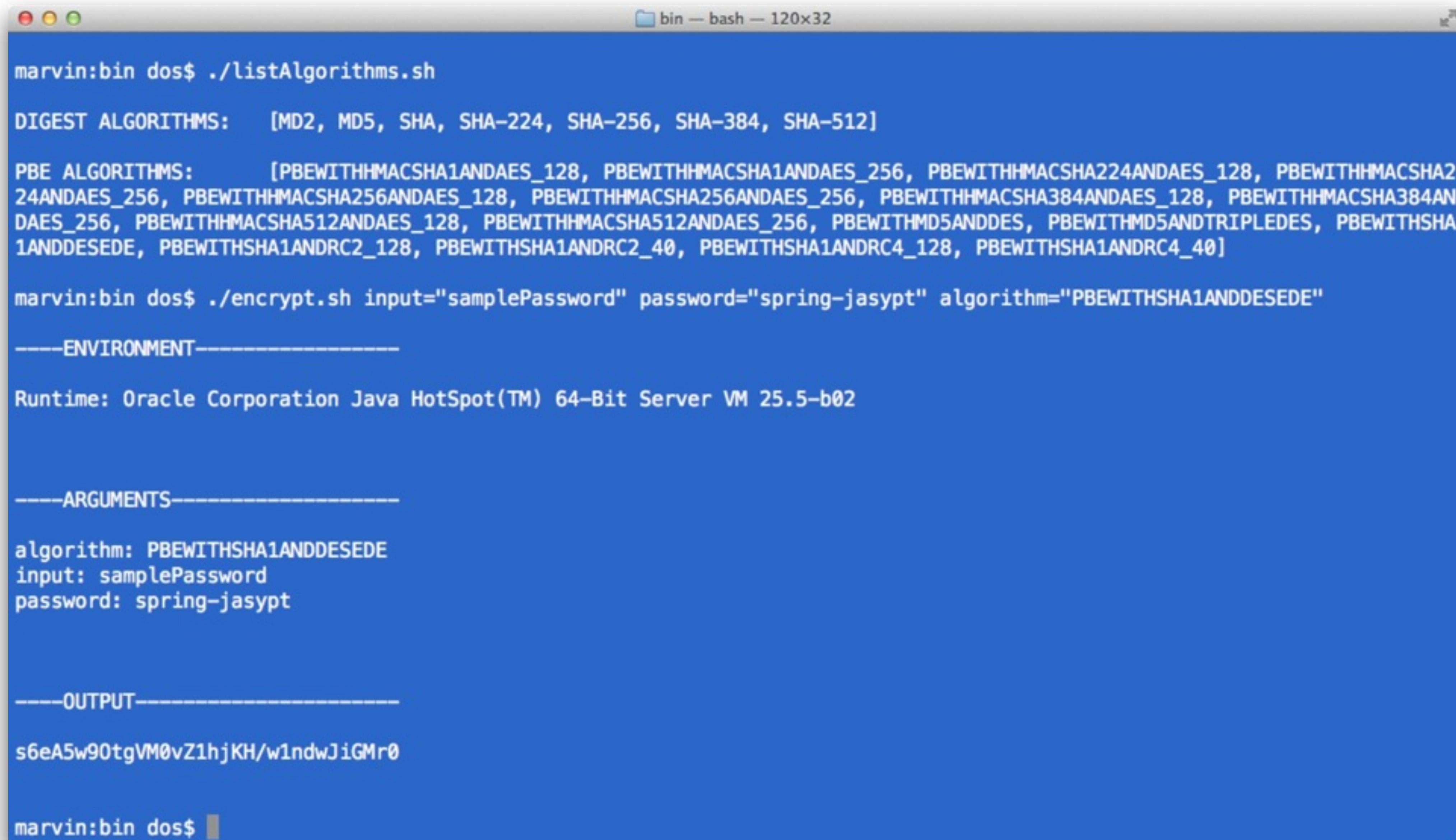
Typical DataSource configuration with Spring

```
<context:property-placeholder  
    location="classpath:spring.properties" />  
  
<bean id="dataSource" class="org...BasicDataSource">  
    <!-- ... -->  
    <property name="url" value="${jdbc.url}" />  
    <property name="username" value="${jdbc.un}" />  
    <property name="password" value="${jdbc.pw}" />  
</bean>
```



```
jdbc.driver=org.hsqldb.jdbcDriver  
jdbc.url=jdbc:hsqldb:mem:sampleDB  
jdbc.un=sampleUser  
jdbc.pw=samplePassword
```

Jasypt command line tools for password generation



The screenshot shows a terminal window titled "bin — bash — 120x32". The output of the terminal is as follows:

```
marvin:bin dos$ ./listAlgorithms.sh
DIGEST ALGORITHMS: [MD2, MD5, SHA, SHA-224, SHA-256, SHA-384, SHA-512]

PBE ALGORITHMS: [PBEWITHHMACSHA1ANDAES_128, PBEWITHHMACSHA1ANDAES_256, PBEWITHHMACSHA224ANDAES_128, PBEWITHHMACSHA24ANDAES_256, PBEWITHHMACSHA256ANDAES_128, PBEWITHHMACSHA256ANDAES_256, PBEWITHHMACSHA384ANDAES_128, PBEWITHHMACSHA384ANDAES_256, PBEWITHHMACSHA512ANDAES_128, PBEWITHHMACSHA512ANDAES_256, PBEWITHMD5ANDDES, PBEWITHMD5ANDTRIPLEDES, PBEWITHSHA1ANDDESEDE, PBEWITHSHA1ANDRC2_128, PBEWITHSHA1ANDRC2_40, PBEWITHSHA1ANDRC4_128, PBEWITHSHA1ANDRC4_40]

marvin:bin dos$ ./encrypt.sh input="samplePassword" password="spring-jasypt" algorithm="PBEWITHSHA1ANDDESEDE"
-----ENVIRONMENT-----
Runtime: Oracle Corporation Java HotSpot(TM) 64-Bit Server VM 25.5-b02

-----ARGUMENTS-----
algorithm: PBEWITHSHA1ANDDESEDE
input: samplePassword
password: spring-jasypt

-----OUTPUT-----
s6eA5w90tgVM0vZ1hjKH/w1ndwJiGMr0

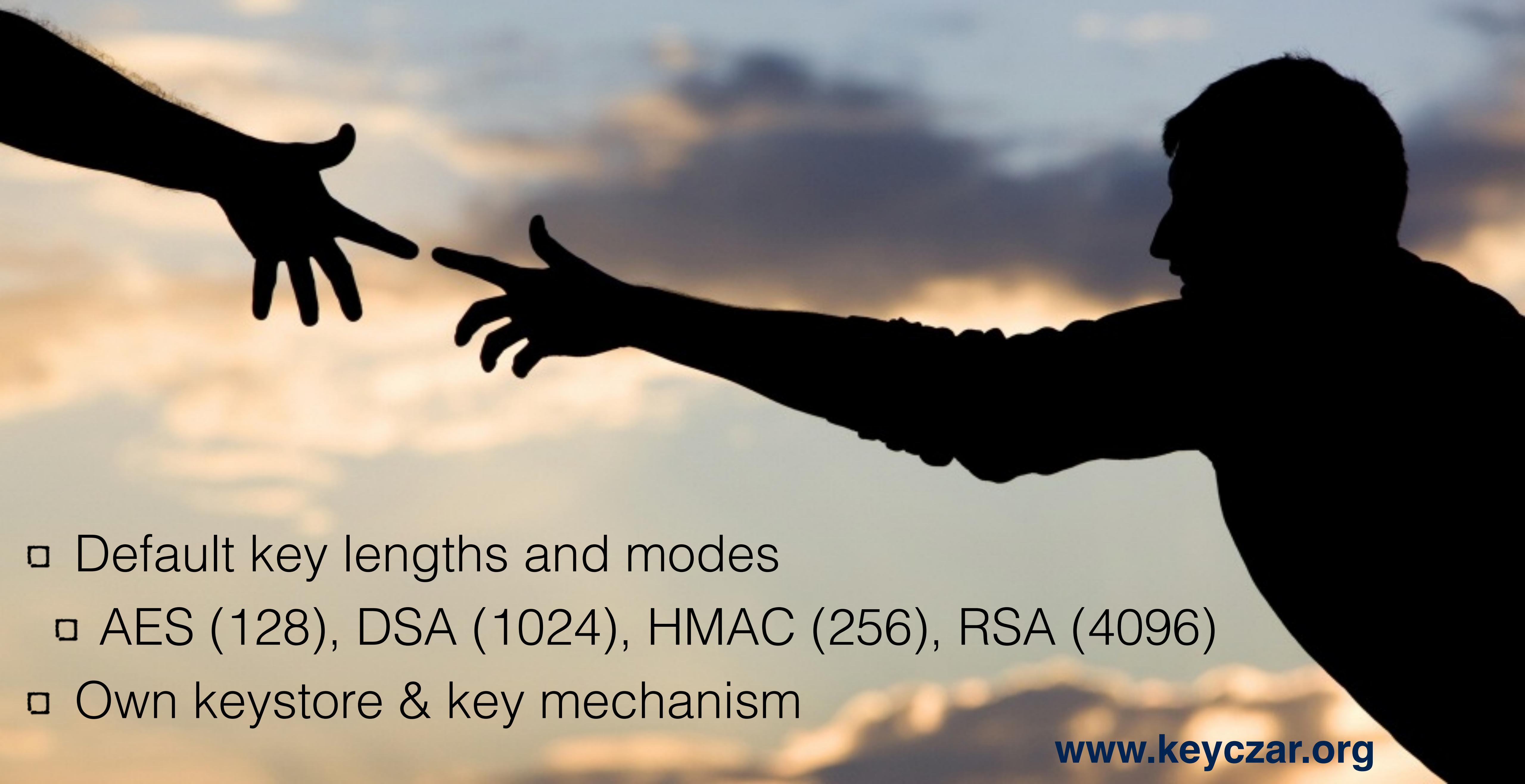
marvin:bin dos$
```

Demo

Encryption & Signatures



Keyczar offers safe algorithms for encrypting and signing



- Default key lengths and modes
 - AES (128), DSA (1024), HMAC (256), RSA (4096)
- Own keystore & key mechanism

Keyczar uses JSON format for key sets and keys



KeyczarTool: java -jar KeyczarTool-0.71g-090613.jar

```
Usage: "KeyczarTool command flags"
Commands: create addkey pubkey promote demote revoke usekey importkey exportkey
Flags: location name size status purpose padding destination version asymmetric crypter pemfile passphrase
Command Usage:
create --location=/path/to/keys --purpose=(crypt|sign) [--name="A name"] [--asymmetric=(dsa|rsa|ec)]
    Creates a new, empty key set in the given location.
    This key set must have a purpose of either "crypt" or "sign"
    and may optionally be given a name. The optional version
    flag will generate a public key set of the given algorithm.
    The "dsa" and "ec" asymmetric values are valid only for sets
    with "sign" purpose.

addkey --location=/path/to/keys [--status=(active|primary)] [--size=size] [--crypter=crypterLocation] [--padding=(OAEP|PKCS)]
    Adds a new key to an existing key set. Optionally
    specify a status, which is active by default. Optionally
    specify a key size in bits. Also optionally specify the
    location of a set of encrypting keys, which will be used to
    encrypt this key set. The optional --padding flag is allowed
    only for key sets created with --version=rsa. If omitted, it
    defaults to OAEP.

pubkey --location=/path/to/keys --destination=/destination
    Extracts public keys from a given key set and writes them
    to the destination. The "pubkey" command Only works for
    key sets that were created with the "--asymmetric" flag.

promote --location=/path/to/keys --version=versionNumber
    Promotes the status of the given key version in the given
    location. Active keys are promoted to primary (which demotes
    any existing primary key to active). Inactive keys are
    promoted to be active.

demote --location=/path/to/keys --version=versionNumber
    Demotes the status of the given key version in the given
    location. Primary keys are demoted to active. Active keys
    are made inactive.

revoke --location=/path/to/keys --version=versionNumber
    Revokes the key of the given version number.
```

Demo

Hashes

A close-up photograph of a person's hand wearing a dark long-sleeved shirt, holding a black smartphone. The phone's screen is visible, showing a login interface. At the top of the screen, the word "login" is written in white lowercase letters. Below it, in large yellow capital letters, is the word "SECURITY ?". Underneath that, the word "password" is written in white lowercase letters. At the bottom of the screen, there is a row of five asterisks ("*****"). The background is dark and out of focus.

login

SECURITY ?

password

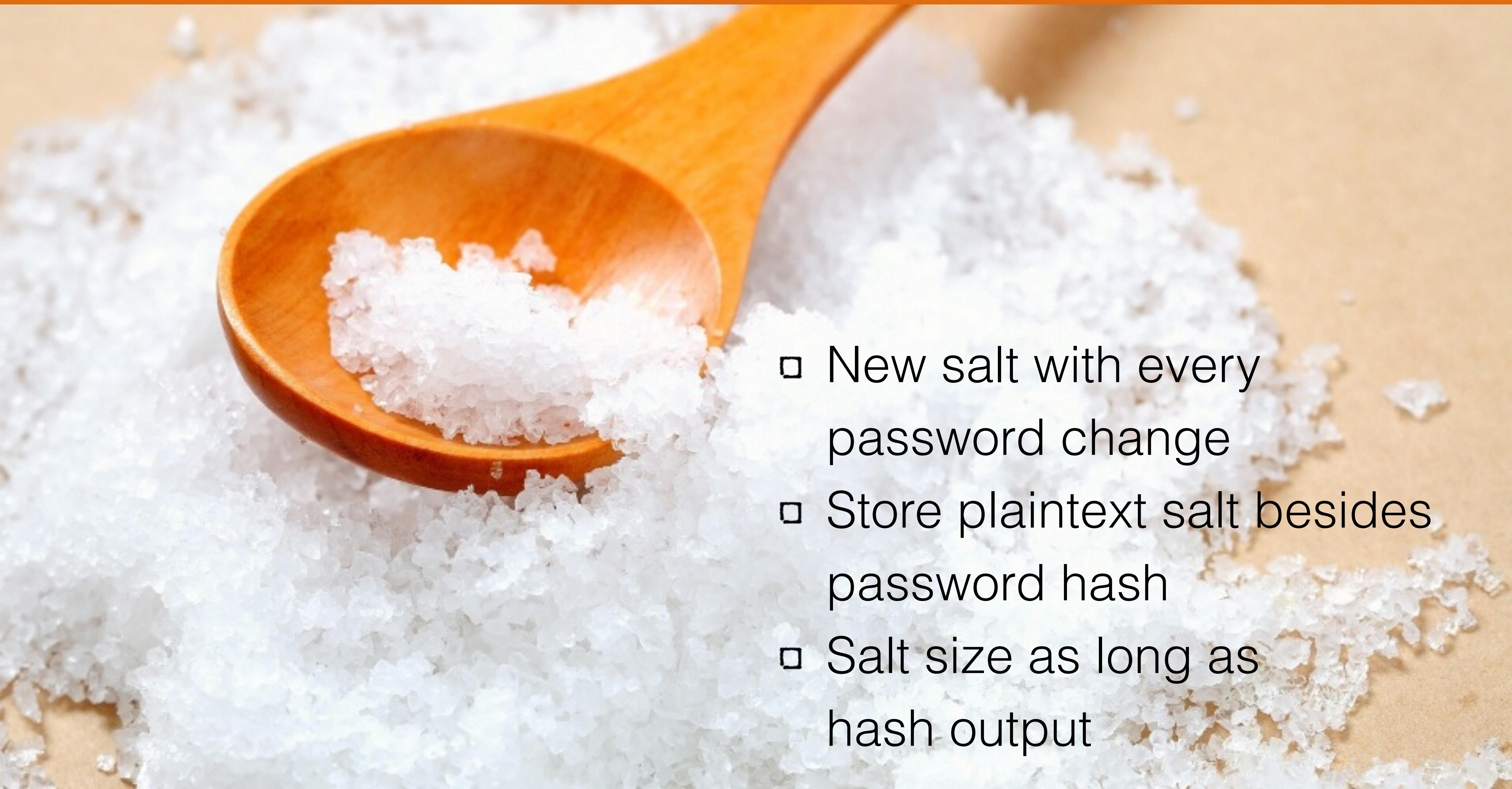
111111 fuck me pass 123456
fuckyou michael ranger hunter buster
football 6969 hockey 1234567 password 12345678 love
jessica mustang master jennifer sunshine 12345678 george
shadow harley qwerty superman dragon
andrew 696969 pussy charlie batman tigger jordan
killer test 1234 12345 michelle
soccer 2000

Hashing is a fast one way operation

- Same input -> same hash value
- Arbitrary long input -> fixed length output



A hashed password without an individual salt is neither safe nor unique



- ❑ New salt with every password change
- ❑ Store plaintext salt besides password hash
- ❑ Salt size as long as hash output

Special password hashing algorithms slow down brute force attacks

1. **PBKDF2** - multiple rounds
2. **scrypt** - resource intensive
3. **bcrypt** - iteration count



Apache Shiro provides safe defaults for encrypting and hashing

- ❑ Multiple iterations configurable
- ❑ Built-in salt generation
 - ❑ Base salt (private)
 - ❑ Public salt

Demo

OWASP Passfault analyzes password strength



www.owasp.org/index.php/OWASP_Passfault

- Don't invent crypto, use it
- Crypto libraries come with limitations but easier usage
- Change your keys frequently



Dominik Schadow dominik.schadow@bridging-it.de
BridgingIT GmbH www.bridging-it.de
Königstraße 42 Blog blog.dominiksshadow.de
70173 Stuttgart Twitter @dschadow

Demo Projects

github.com/dschadow/JavaSecurity

Cryptographic Storage Cheat Sheet

www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

CrypTool

www.cryptool.org

Pictures

www.dreamstime.com

